A Simple Virtual Machine for Multi-Agent Systems Based On Cellular Coordination Mechanisms

Chris Darken

October 29, 2002

KEY CHARACTERISTICS OF CELLULAR COMM.

- There are a limited number of discrete signals
- However, any given signal may produce very complicated behavior in the recipient, in particular, one signal may cascade into many
- Signals and receptors may be blocked by causing them to bind to other than their intended target

MOTIVATION

- Rigorously describe an ticket/connector-style MAS architecture that is as simple as possible, but still potentially useful

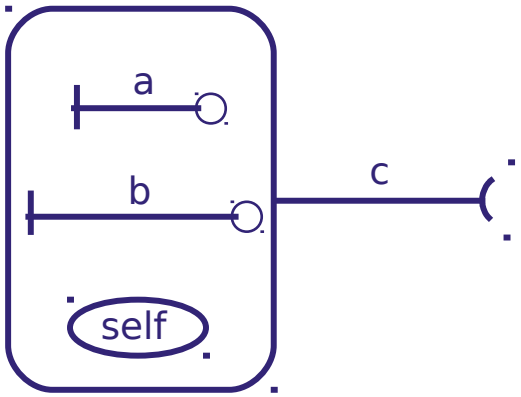
OUTLINE

- Presentation of the MAS VM
  - Ticket structure
  - Computation mechanism
- Simple example
- Proof of Turing equivalence
- List of possible extensions

SALIENT FEATURES

Signal of type a

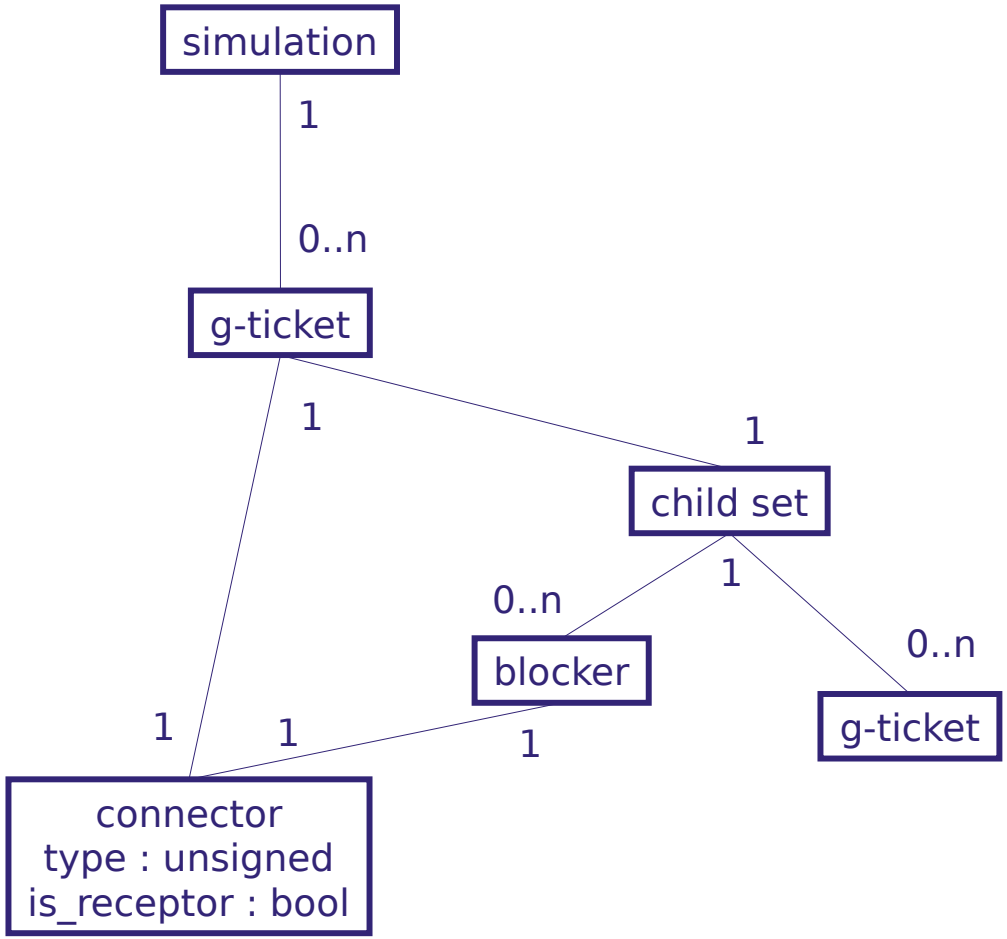
Blocker of type a



Self-perpetuating receptor of type c that also produces a and b signals

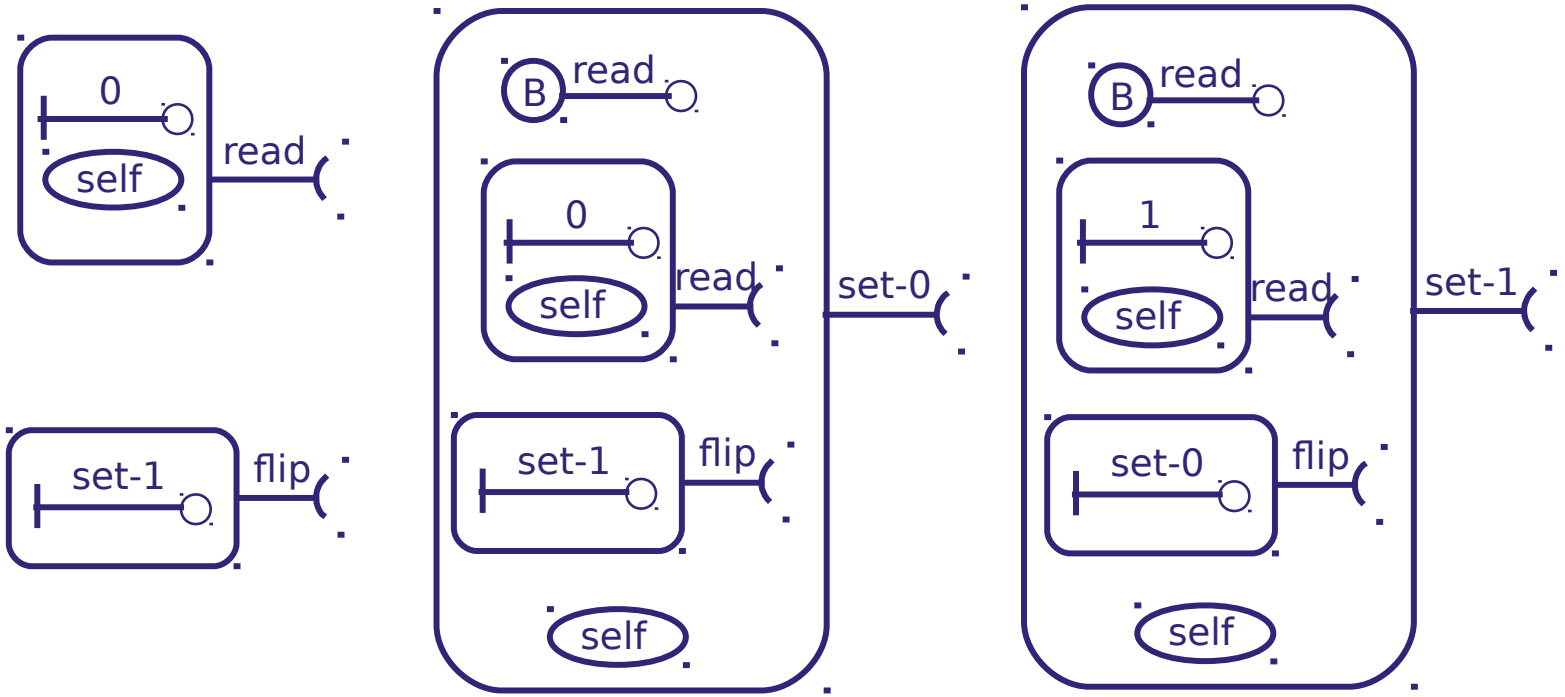
- Only intra-agent scope considered (easy to add multiple agent scopes)
- Only one action is possible when a ticket is activated: the creation of tickets
- General computational actions inside tickets are not allowed
- Ticket creation happens simultaneously (I.e. not in sequence)
- Tickets can create themselves
- Blocking is modeled as instantaneous ticket destruction
- Some example tickets are at left

ENTITY RELATIONSHIP DIAGRAM



MAS EXAMPLE: MULTI-READ FLIP-FLOP

Starts in 0 state



THE MAS VIRTUAL MACHINE

- The *state* of the simulation is the multi-set of top-level g-tickets (I.e. g-tickets in a child set are not in the state)
- Given a simulation state, the *matching set* is the multi-set consisting of all pairs of g-tickets in the state such that
  - The two g-tickets in each pair have the same type (I.e. gt1.type == gt2.type)
  - One of the g-tickets is a receptor, and the other is not (I.e. either gt1.is_receptor==true and gt2.is_receptor==false, or vice versa)
- At each computational step, an element of the matching set is selected. How this element is selected is intentionally not specified, but it might be random selection, for instance. If the matching set is empty, computation terminates.
- After pair selection, the state is updated as follows:
  - Both g-tickets in the pair are removed from the state
  - All g-tickets in the state that bind with any blocker from either g-ticket are immediately removed from the state.
  - All g-tickets in the child sets of either g-ticket in the pair are added to the state

SEMANTICS

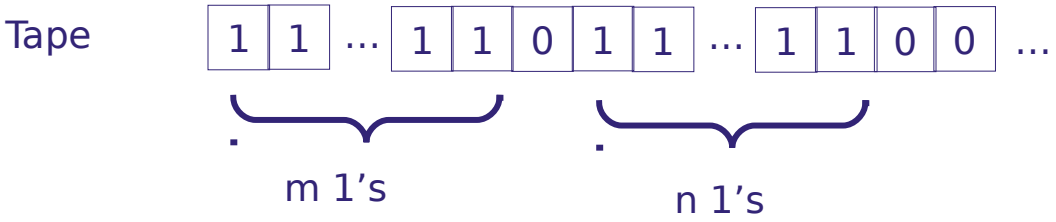
- An outcome of a run can be complete characterized by the sequence of states the virtual machine goes through
- The complete semantics (meaning) of a state, then, may be taken to be the set of possible outcomes (I.e. the set of outcomes consistent with the state update rules)

TURING MACHINE: INTRODUCTION

- Consists of a tape, a read/write tape head, and an automaton
- The tape is a sequence of 1’s and 0’s
- For programs that can be run on an actual computer, the tape is finite
- The tape head is positioned over one specific entry in the tape
- The automaton is always in one of a finite set of “states”. At each computational step, the automaton reads the current bit via the tape head and depending on its value and the automaton’s state, it…
  - Writes the current bit to either 0 or 1
  - Moves the tape head either up or down the tape
  - Transitions to any state

TURING MACHINE: EXAMPLE

- Computes m+n, where m and n are represented as strings of m and n 1's separated by a zero and the result is represented in similar form



	0	1
1	1U2	1U1
2	0D3	1U2
3		0U4

bit read from tape

state

write

move

new state

TURING MACHINE: MATHEMATICAL MODEL

- The tape can be represented as a function t:**N**→{0,1}
- The tape head can be represented by its position, an element of **N**
- The automaton can be represented by a state (a member of the finite set **S**={S1.S2….,Sn}) and the following three functions
  - w : **S**X{0,1}→{0,1} The write function
  - m : **S**X{0,1}→{up,down} The move function
  - s : **S**X{0,1}→**S** The new state function
- The state of the entire Turing machine consists of:
  - The state of the automaton
  - The complete contents of the tape
  - The current tape position

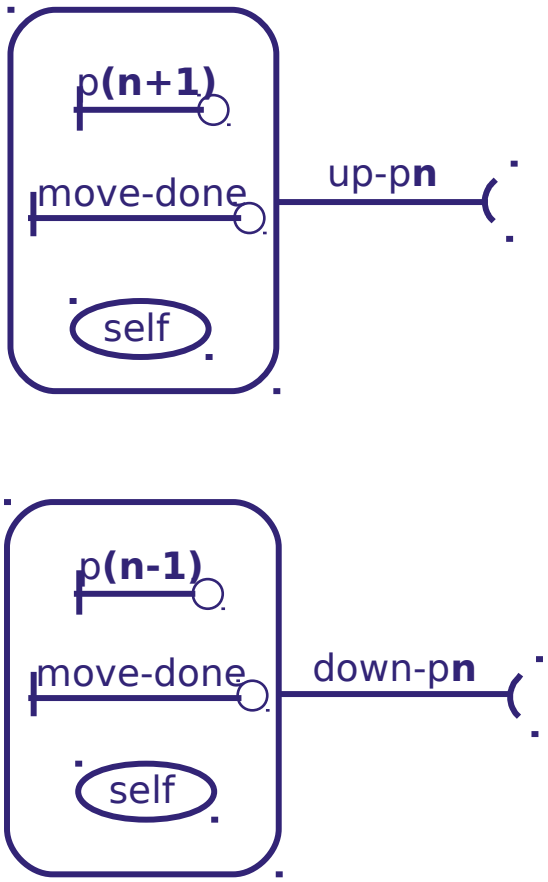
TURING EQUIVALENCE

- It is pretty obvious that we can simulate any instantiation of our MAS VM on a conventional computer (I.e. a Turing machine)
- But can we simulate any Turing machine on our MAS VM?
- We will prove that we can with a “constructive proof”, I.e we will give a recipe to build an MAS VM state that will perform the same computation as any given Turing machine

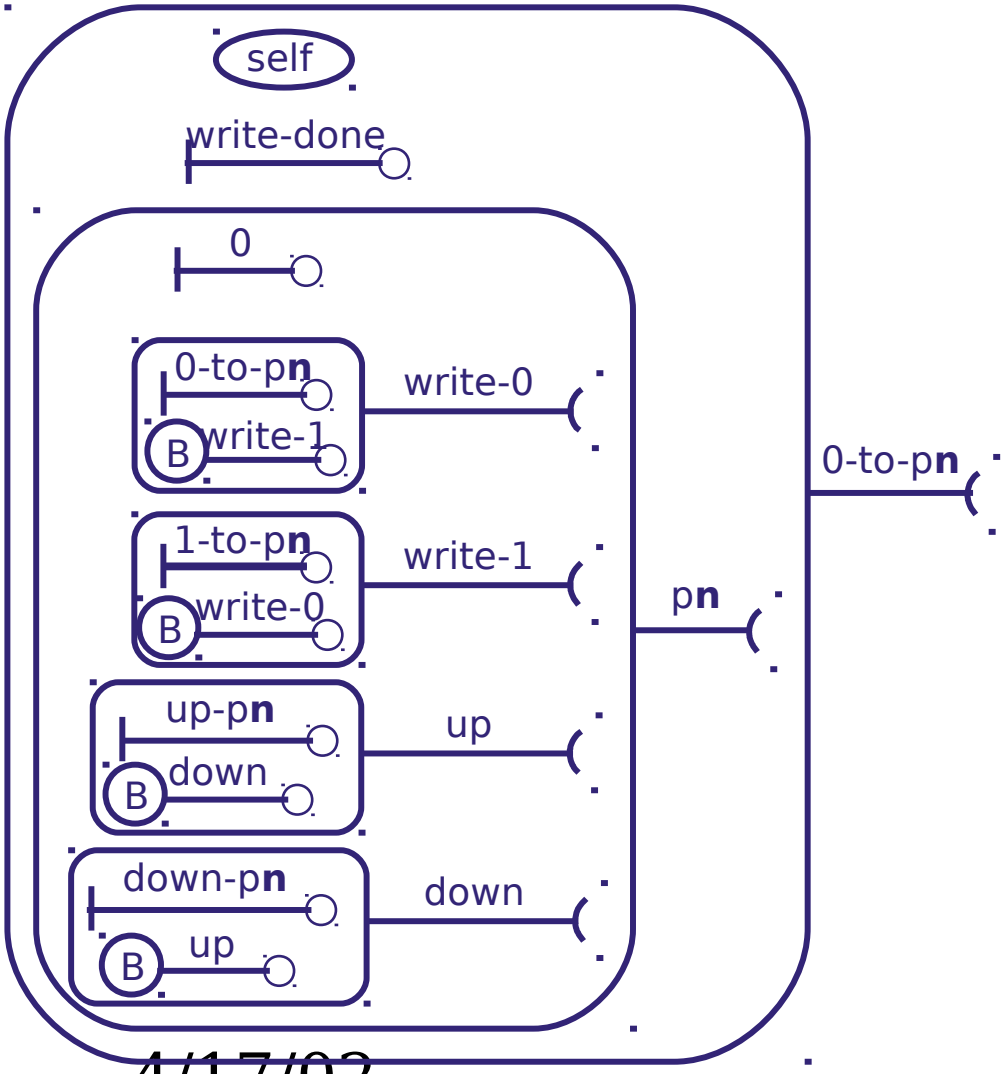
MAS MACHINE: PERSISTENT KNOWLEDGE

- Ordering of tape positions (e.g. “up” from 5 is 6 and “down” from 5 is 4)
- How to set each tape position to 0 or 1
- The values of w, m, and s for each state

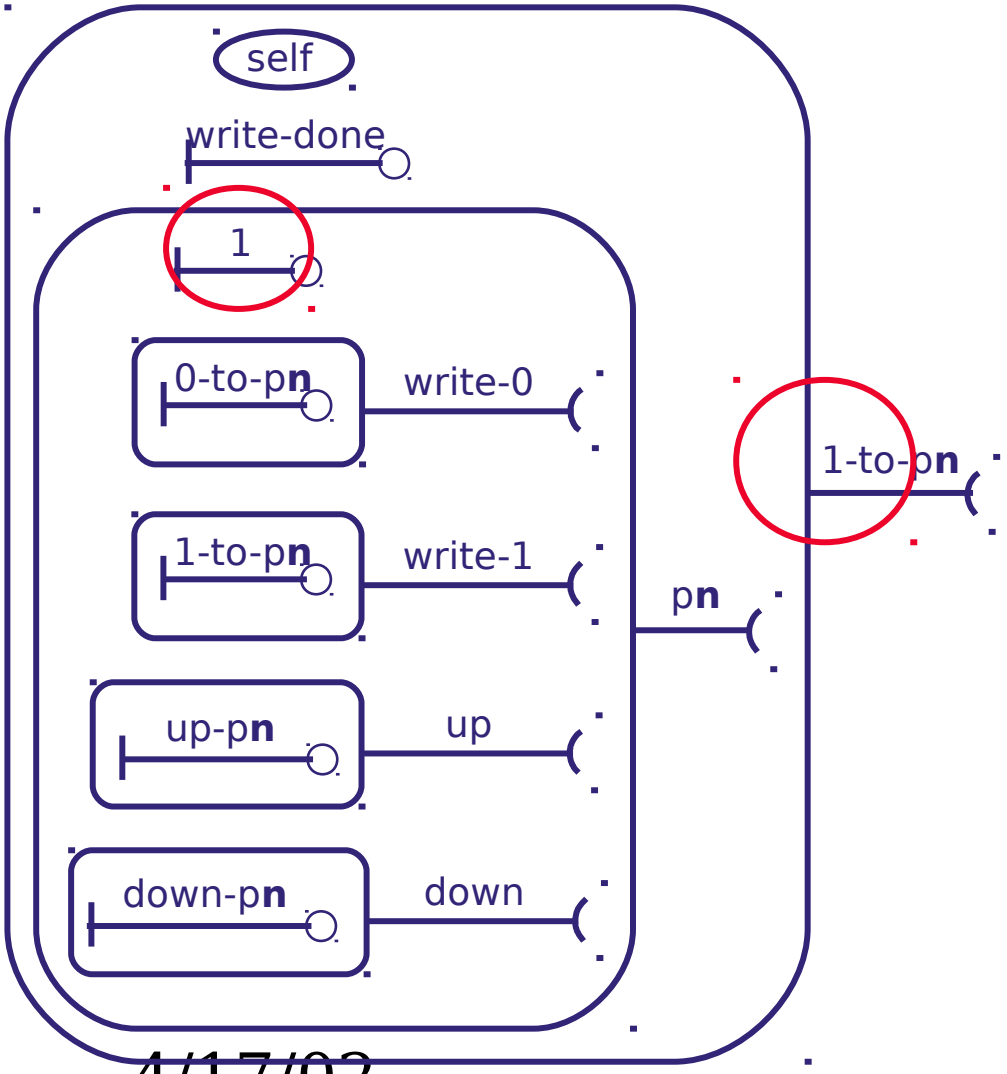
TAPE POSITION SEQUENCE



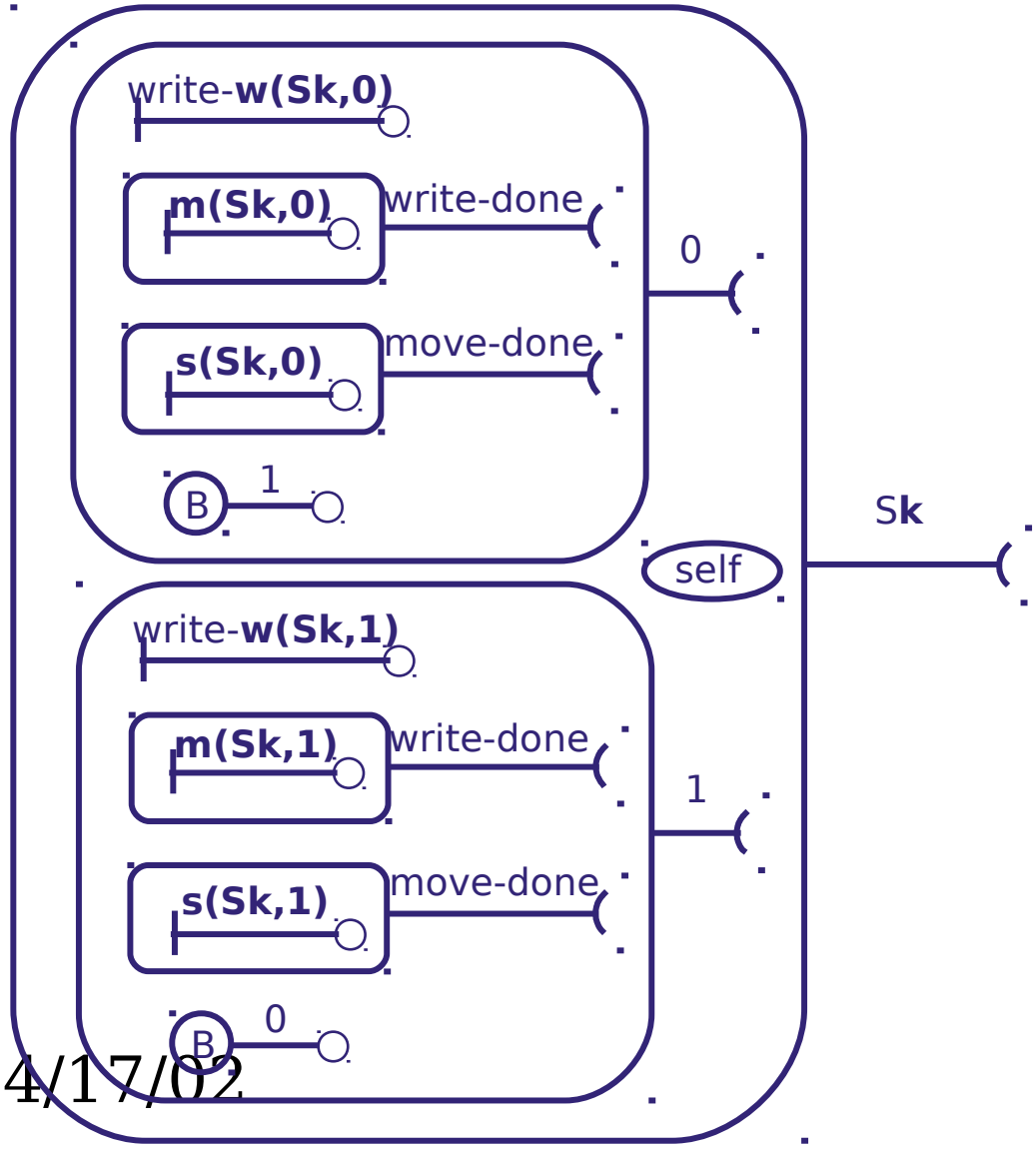
CHANGING TAPE VALUES



CHANGING TAPE VALUES



STATES


write-w(Sk,0)
m(Sk,0)
write-done
s(Sk,0)
move-done
B
1
0
Sk
self
write-w(Sk,1)
m(Sk,1)
write-done
s(Sk,1)
move-done
B
0
1

PUTTING THE MAS VM INTO CORRESPONDENCE WITH AN ARBITRARY TURING MACHINE STATE

- Add all persistent knowledge g-tickets to state. Note that the state g-tickets are specific to the Turing machine we will emulate.
- Set tape contents, automaton state, and tape position
  - Tape contents: for each n, add 0-to-p**n** or 1-to-p**n**.
  - Add (B)-write-done-C
  - Tape position: if r is the initial tape position, add p**r**
  - After each g-ticket is added, run the VM to termination
  - Automaton state: if the desired state is q, add S**q**

MACHINE STATE ISOMORPHISM

- The tape values can be read off as the 0-O or 1-O in the p**n**-C
- The current tape position is the **n** in the 0-to-p**n** inside the write-0-C
- The current state is the S**k** of the S**k**-C

COMPLETE MAS VM STATE

- Initial connectors:
  - Persistent knowledge
    - up-p**n**-C, down-p**n**-C
    - 0-to-p**n**-C, 1-to-p**n**-C
    - S**q**-C
  - Transient knowledge
    - p**n**-C
    - 0-O OR 1-O
    - write-0-C, write-1-C
    - up-C, down-C

PROOF BY INDUCTION

- By the state-setting process described before, the initial state of the MAS VM can be made isomorphic to the initial state of the Turing machine
- We need to show additionally that for an arbitrary state of the Turing machine, if the MAS VM is isomorphic to that state, and both machines advance one computational step, the resulting states are isomorphic
- Let **X** be the lexical variable for the bit in the current tape position
- Let $\underline{\mathbf{X}}$ be 1-**X**

PROOF

- {S**q**-O, S**q**-C}
  - Add 0-C, 1-C
- {**X**-O, **X**-C}
  - Del **X**-C
  - Add write-**w(Sq,X)**-O, write-done-C, move-done-C
- {write-**w(Sq,X)**-O, write-**w(Sq,X)**-C}
  - Del write-**w(Sq,X)**-C
  - Add **w(Sq,X)**-to-p**r**-O
- {**w(Sq,X)**-to-p**r**-O, **w(Sq,X)**-to-p**r**-C}
  - Add **w(Sq,X)**-to-p**r**-C, write-done-O, p**r**-C
- {write-done-O, write-done-C}
  - Add **m(Sq,X)**-O
- {**m(Sq,X)**-O, **m(Sq,X)**-C}
  - Del **m(Sq,X)**-C
  - Add **m(Sq,X)**-p**r**-O
- {**m(Sq,X)**-p**r**-O, **m(Sq,X)**-p**r**-C}
  - Add move-done-O, **m(Sq,X)**-p**r**-C
  - If **m(Sq,X)** is up, **Op**=+, otherwise **Op**=-. p(**rOp1**)-C is added.
- {p**(rOp1)**-O, p**(rOp1)**-C, move-done-C, move-done-O}
  - By consideration of both cases, it can be shown that the outcome does not depend in any way on which pair is selected. We will choose the first pair.
  - Let **Y** be the bit stored at tape position **rOp1** during initialization. Add **Y**-O, write-0-C, write-1-C, up-C, down-C.
- {move-done-C, move-done-O}
  - Add **s(Sk,X)**-O
- At this point, we observe that the MAS VM has been placed in a state isomorphic to the Turing machine successor state, completing the proof.

SIMPLIFICATIONS (OMISSIONS)

- Groups of connectors (and/or flag and how they match)
  - Universal key
  - Universal responder
- Connectors with limited (intra-agent vs. extra-agent) scope
- Time (finite life of tickets, activation times)
- Space (distances among agents, diffusion time)
- Action sets
  - Sequences
  - Any-order lists
  - Do-any-one lists
- Other binding semantics
  - Passing information packets
  - Controlling method scope

FUTURE WORK

- What is the minimal ticket-based VM that is Turing equivalent?
  - Tickets with signal-gendered connectors need not do ticket creation
  - Blockers can be restricted to signal-gendered connectors
  - Can we do without blockers altogether?
  - Currently the number of connector types scales linearly with the required tape size. Can this be avoided?
- How can formalisms including general computational actions be rigorously described?